The design of safe automotive embedded systems - Some problems, solutions and open issues

Françoise Simonot-Lion

The design of safe automotive embedded systems - Some problems, solutions and open issues

Françoise Simonot-Lion, LORIA UMR 7503, Nancy Université, France (simonot@loria.fr)

***Abstract*—From the last decade, the number of software based systems embedded in a car increases every year. The reasons for this evolution are economical as well as technological. On the one hand, this situation is the result of the decreasing cost of hardware components, their increasing reliability and performances and the emergence of embedded fieldbuses; on the other hand, software technology makes easier and less costly the introduction of new functions. Formerly confined to functionalities such as engine or chassis control, this evolution now affects all car domains: wipers, door controls, lights, air condition, braking assistance, multimedia, etc. In the future, even critical functions, as for example, braking or steering, will be fully controlled by electronic systems leading to the X-by-Wire concept. The realization of such systems is obtained through a complex cooperative development process shared by several actors, in particular, OEM (carmakers) and tier-1 suppliers. Furthermore, it's no longer possible to study each system as a stand-alone one and all the partners involved in the design of these systems have to observe a global and common view of the whole embedded architecture. In this context, the main challenge is nowadays to provide means for an efficient development of a safe and optimal embedded system. In this presentation, we will focus on some keywords whose impact and meaning may look antagonist. For example, component, modularity and reusability are recurrent concepts aiming to increase the efficiency of a development while reducing its length. Nevertheless, these principles can be opposed to safety, reliability, dependability purposes. Indeed, the verification of these required properties have to be done on the whole system and not only on a single component. Therefore, we have to complete these first concepts and to introduce the notion of composition of components and moreover of interoperability of components. We will show how this composition can be described through a reference model of embedded architecture that provides on the one hand a standard embedded middleware and on the other hand, an architecture description language. Then, we will focus on the verification of safety/dependability properties and identify which kind of activities they can require and how these activities are related to the first point.**

I. General context

In-vehicle embedded systems are usually classified according to domains that correspond to different functionalities, constraints and models [**?**], [**?**], [**?**]. They can be divided among "vehicle centric" functional domains, such as powertrain control, chassis control, active or passive safety systems and "passenger centric", where multimedia/telematics, body/comfort and HMI (Human Machine Interface) can be identified.

Carmakers distinguish several domains for embedded electronics in a car, even though sometimes the membership of only one domain for a given comportment is not easy to justify. According to the glossary of the European ITEA EAST-EEA project [**?**], a domain is defined as "a sphere of knowledge, influence, and activity in which one or more systems are to be dealt with (e.g. are to be built)". The term domain can be used as a means to group mechanical and electronic systems.

Historically five domains were identified: Powertrain, Chassis, Body, Telematics and Human Man Interface (HMI). The Powertrain domain is related to the systems which participate in the longitudinal propulsion of the vehicle, including engine, transmission and all subsidiary components. The Chassis domain refers to the four wheels and their relative position and movement; in this domain, the systems are mainly steering and braking. According to EAST-EEA definition, the Body domain includes the entities which do not belong to the vehicle dynamics, thus being those which support the car's user, such as airbag, wiper, lighting, window lifter, air conditioning, seat equipment, etc. The HMI domain includes the equipment allowing information exchange between electronic systems and the driver (displays and switches). Finally, the Telematic domain is related to components allowing information exchange between the vehicle and the outside world (radio, navigation system, internet access, payment).

From one domain to another, the electronic systems often have very different features. For example, the Powertrain and Chassis domains both exhibit hard real-time constraints and a need for high computation power. However, the hardware architecture in the Chassis domain is more widely distributed in the vehicle. The Telematic domain presents requirements for high data throughput. From this standpoint, the technological solutions used are very different, for example, for the communication networks, but also for the design techniques and verification of the embedded software.

II. Standardized components, models and processes

As pointed out in section II-D, the design of new in-vehicle embedded systems is based on a cooperative development process. Therefore, it must ensure the interoperability between components developed by different partners and ease their portability onto various platforms in order to increase the system's flexibility. On the one hand, a means to reach these objectives is furnished by the standardisation of services sharing hardware resources between application processes, in particular, networks and their protocols and operating systems. On the other hand, portability is achieved through the specification of a common middleware. Notice that such a middleware also has to deal with the interoperability properties.

Finally, a standardized and common support for modelling and documenting systems all along their development eases the design process itself and, more specifically, the exchanges between the different partners at each step of the development cycle. In the following we introduce some of the standardized components or models aiming to support this cooperative development process.

A. In-vehicle networks and protocols

Specific communication protocol and networks have been developed to fulfill the needs of automotive embedded systems. In 1993, the "SAE Vehicle Network for Multiplexing and Data Communications Standards Committee" identified 3 kinds of communication protocols for in-vehicle embedded systems based on network speed and functions[**?**]; they are called, respectively, "class A", "class B" and "class C". The same committee also published a requirement list concerning safety critical applications. In particular, the communication protocol for X-by-Wire systems must respect requirements for "dependability and fault-tolerance" as defined for the class C [**?**]. Networks compliant to class A provide a bit rate below 10 Kbits/second and are dedicated to sensor and actuator networks; the LIN bus and TTPI/A bus are among the most important protocols in this class. Class B specifies a medium speed (10K b/s to 500K b/s) and is, thus, convenient for transferring information in vehicle centric domains and the body's electronics systems. CAN-B is a widely used class B protocol. Class C has been defined for safety-relevant systems in powertrain or chassis domains. The data rates here are lower than 1 Mbit/s. CAN-C (high speed CAN), TTP/C and FlexRay are fall into this category. They have to provide highly reliable and fault tolerant communication. Obviously, class C networks will be required in future X-by-wire applications for steering and braking. For further information on automotive embedded networks, the reader can refer to[**?**], [**?**].

B. Operating Systems

OSEK/VDX (Offene Systeme und deren schnittstellen für die Elektronik im Kraft-fahrzeug) [**?**] is a multi-task operating system that is becoming a standard in European automotive industry. This standard is divided in four parts: OSEK/VDX OS is the specification of the kernel itself; OSEK/VDX COM concerns the communication between tasks (internal or external to an ECU); OSEK/VDX NM addresses network management; and finally, OSEK/VDX OIL is the language that supports the description of all the components of an application. Certain OSEK-targeted applications are subject to hard real-time constraints, so the application objects supported by OSEK have to be configured statically.

OSEK/VDX OS provides services on objects like tasks ("basic tasks", without blocking point, and "extended tasks", that can include blocking points), events, resources and alarms. It proposes a Fixed Priority (FP) scheduling policy which is applied to tasks that can be preemptive or non preemptive, and combined with a reduced version of the Priority Ceiling Protocol (PCP) [**?**], [**?**] in order to avoid priority inversion or deadlock due to exclusive resource access. Inter-task synchronisation is achieved through private events and alarms. The implementation of an OSEK/VDX specification has to be compliant to one of the four conformance classes -BCC1, BCC2, ECCI, ECC2 - which are specified according to the supported tasks (basic only or basic and extended), the number of tasks on each priority level (only one or possibly several), and the constraints of the reactivation counter (only one or possibly several). BBC1 defines a restricted implementation whose aim is to minimize the size of the corresponding memory footprint, the size of the data structures and the complexity of the management algorithms. ECC2 specifies the implementation of all the services. The MODISTARC project (Methods and tools for the validation of OSEK/VDX based DISTributed ARChitectures) [**?**] provided the relevant test methods and tools to assess the compliance of OSEK/VDX implementations.

In order to describe an application configuration, the OSEK consortium provided a specific language, called OSEK/OIL (OSEK Implementation Language). This language allows, for one ECU, the description of several application configurations, called application modes. For example, the application configurations can be specified for a normal operation mode, for a diagnosis mode and for a download mode.

The dependability purpose and fault tolerance for critical applications is usually achieved by a Time-Triggered approach [**?**]. So, the Time-Triggered operating system OSEKtime [**?**] was defined. It supports static and time-triggered scheduling, and offers interrupt handling, dispatching, system time and clock synchronisation, local message handling, and error detection mechanisms. Thanks to these services, an application running on OSEKtime can be predictable. OSEKtime is compatible to OSEK/VDX and is completed by FTCom layer (Fault Tolerant Communication) for communication services. It should be noted that the specification of the basic software for AUTOSAR is based on services from OSEK and OSEKtime.

Rubus is another operating system tailored for the automotive industry and used by Volvo Construction Equipment. It was developed by Arcticus systems [**?**]. Rubus OS is composed of three parts: the Red Kernel, which manages the execution of off-line scheduled time-triggered tasks, the Blue Kernel, which is dedicated to the execution of event-triggered tasks, and the Green Kernel, which is in charge of external interrupts. As for OSEK/VDX OS, the configuration of the tasks has to be defined statically off-line.

For multimedia and telematics applications, the operating systems are generic ones, such as VxWorks (from WindRiver) or even a Java machine. "Microsoft Windows Automotive 5.0" extends the classical operating system Windows CE with telematic-oriented features and was, for example, installed among other in certain Citroën Xsara and BMW 7 Series.

C. Middleware

Flexibility and portability of applicative components require two main features. On the one hand, an application embedded on a distributed platform is based on the description of

elements, the semantics of the interaction types among the elements and, consequently, the semantics of their composition. Note that these interactions must be specified disregarding the allocation of components on an ECU. On the other hand, the properties required at the application level, mainly timing and dependability properties, must be met when components are allocated onto a technical platform (operating systems, communication drivers and protocol, input/output drivers, etc.). Traditionally, these features are achieved through the specification of a middleware. Firstly, with the structure of the middleware, i.e. the elementary software components allocated on each ECU and the way they interact has to be formally identified and, secondly, the interface services that furnish a way for applicative components to use the middleware services independently of their allocation have to be furnished. During the last decade, several projects focused on this purpose. See, for example, the German Titus project [**?**] started by DaimlerChrysler in 1994. The purpose of this project was to develop an interface-based approach similar to the ROOM methodology [**?**], but differing considerably in certain details, mainly in making an "actor-oriented" approach which was suitable for ECU software. The French EEA project [**?**] identified the classes of software components implemented on a ECU. Then the European ITEA EAST EEA project combined these classes and proposed a more advanced architectural view of an ECU [**?**]. The mission of the DECOS project, supported by the 6th EU Framework Programme, was to develop an architecture-based design methodology, to identify and specify the associated COTS hardware and software components, and to provide certified development tools and advanced hybrid control technologies. This project targeted control systems concerning the dependability of software-intensive systems, in particular, in avionics (Airbus) and automotive industries. After that, the Volcano project concentrated on just the communication services and provided a way (both middleware components and interface services) for supporting the signal exchanges between distant applicative components by hiding the underlying protocol. Volcano targeted the timing properties imposed on signal exchanges [**?**], [**?**].

Finally, the AUTOSAR consortium (AUTomotive Open Standard ARchitecture) standardized a common software infrastructure for automotive systems [**?**]. Once put into practice, it will bring meaningful progress when designing an embedded electronic architecture because: 1 - it will allow the portability of the functions on the architecture and reuse them; 2 - it will allow the use of hardware Components Off-the-Shelf (COTS); 3 - on a same ECU it will be able to integrate functions from different suppliers. During the lifetime of the car, this standard will facilitate updating the embedded software as this technology evolves, as well as the maintenance for the computers.

D. Architecture Description Languages for automotive applications

Sharing the same modelling language between the different partners involved in the design of these in-vehicle embedded systems is a means to support an efficient collaborative development process. In this context, such a language will have to allow for describing a system at different steps of its development (requirement specification, functional specification, design, implementation, tuning, etc.) by taking into consideration the different viewpoints of the actors as well as ensuring a consistency between these different views. It will also need to reflect the structure of the embedded systems as an architecture of components (hardware components, functional components, software components). The concept of Architecture Description Languages (ADLs), developed for large software applications [**?**], is well suited to these objectives. ADLs are used to describe the structure of a system by means of the components which are interconnected in a way to form configurations. These descriptions are free of implementation details, one of the objectives being the mastery of the structure of complex systems. So, the composition (associated to hierarchy) used to specify the assembly of the elements constitutes the fundamental construction. For critical systems, as it is the case in automotive electronics, an ADL must support not only the specification of the functional aspects of the system, but also those that are extra-functional (timing properties, dependability properties, safety properties), and other transformation and verification facilities between design and implementation, while maintaining a consistency between the different models. In 1991, Honeywell Labs specified an ADL, MetaH [**?**] that was dedicated to avionics systems. This language was chosen in 2001 to be the core of an Avionics Architecture Description Language (AADL) standard under the SAE authority [**?**]. For the specific automotive domain, several languages were proposed. For example, the language EAST-ADL [**?**], which is tightly related to the generic reference architecture mentioned in the previous section, was specified in the European project ITEA EAST-EEA project [**?**] and extended in the ATESST project [**?**]. The purpose of EAST-ADL is to provide support for the non-ambiguous description of in-car embedded electronic systems at each level of their development. It provides a framework for modeling such systems through 5 abstraction levels, divided into 7 layers (also called "artifacts"), as shown in Figure 1. Some of these layers are mainly concerned with software development while others are linked to the execution platform (ECUs, networks, Operating Systems, I/0 drivers, middleware, etc.). All these layers are tightly linked, allowing traceability among the different entities that are implicated in the development process. Besides the structural decomposition, which is typical for any software development or modeling approach, the EAST ADL also has means for modeling cross-cutting concerns such as requirements, behavioral description and validation, and verification activities. At vehicle level, the *Vehicle Feature Model* describes the set of user-visible features. Examples of such features are anti-lock braking or windscreen wipers. The *Functional Analysis Architecture*, at the analysis level, is an artifact that represents the functions that realize the features, their behavior and their cooperation. There is an n-to-n mapping between Vehicle Feature Model and Functional Analysis Architecture entities, i.e. one or several functions may realize one or several features. The *Functional Design Architecture* (design level) models a decomposition or refine-

ment of the functions described at analysis level in order to meet constraints regarding allocation, efficiency, re-use, supplier concerns, etc. Again, there is an n-to-n mapping between the entities for Functional Design Architecture and the corresponding ones in Functional Analysis Architecture. At the implementation level, the role of the *Function Instance Model* is to prepare the allocation of software components and exchanged signals to OS tasks and frames. It is, in fact, a flat software structure where the Functional Design Architecture entities have been instantiated. It provides an abstraction of the software components to implement. In order to model the implementation of a system, EAST-ADL furnishes, on the one hand, a way to describe the hardware platforms and their available services (operating system, protocol, middleware) and, on the other hand, a support for the specification of how a Function Instance Model is distributed onto a platform. This is done thanks to three other artifacts. The *Hardware Architecture* includes the description of the ECUs and, more precisely, those for the micro-controller used, the sensors and actuators, the communication links (serial links, networks) and their connections. The *Platform Model* defines the operating system and/or Middleware API and, in particular, the services provided (schedulers, frame packing, memory management, I/0 drivers, diagnosis software, download software etc.). Finally, the *Allocation Model* is used at the operational level. It models the tasks, which are managed by the operating systems and frames, which are in turn managed by the protocol. This is the result of the Function Instance Model entities being mapped onto the Platform Model. Note that the specification of a Hardware Architecture and a Platform Model is done simultaneously with function and software specification and can even be achieved during the definition of an allocation model. At this lowest abstraction level, all of the implementation details are captured. The EAST-ADL language provides consistency within and between the artifacts belonging to the different levels, from a syntactic and semantic point of view. This makes an EAST-ADL based model a strong and non-ambiguous support, not only for the realization of software components, but also for building, automatically, models which are suited for format validation and verification activities [**?**], [**?**].

III. The certification issue of safety-critical in-vehicle embedded systems

Several domains are recognized as critical, for example, nuclear plants, railways, avionics. They are subject to strong regulations and must prove that they meet rigorous safety requirements. Therefore, the manner of specification and the management of the dependability / safety properties represent an important issue, as well as the certification process. This problem has become of primary importance for the automotive industry due to the increasing number of computer-based systems, such as critical functions like steering and braking-Consequently, several proposals have been under study. The existing certification standards [**?**], ARP 4754 [**?**], RTCA/DO-178B [**?**] (used in avionics), or EN 50128 [**?**] (applied in the railway industry), provide stringent guidelines for the development of a safety-critical embedded system. However, these standards are hardly transposable for in-vehicle software-based systems: partitioning of software (critical / non critical), multiple versions, dissimilar software components, use of active redundancy, and hardware redundancy. In the automotive sector, the Motor Industry Software Reliability Association (MISRA), a consortium of the major actors for automotive products in the UK, proposes a loose model for the safety-directed development of vehicles with on-board software [**?**]. Also, the generic standard IEC 61508 [**?**], used for Electrical / Electronic / Programmable Electronic systems appears to be a good candidate for supporting a certification process in the automotive industry. Finally, an upcoming standard is being developed, derived from that for the IEC, which serves automotive-specific needs.

The ISO international draft standard ISO WD 26262, planned for 2008, is currently under progress in cooperation with the EU, the USA and Japan [**?**], [**?**].

The next step will consist in the assessment of its usability by the members of the ISO association. The ISO WD 26262 standard is applied to functional safety, whose purpose is to minimize the danger that could be caused by a possibly faulty system. The ISO draft specifies that functional safety is ensured when "*... a vehicle function does not cause any intolerable endangering states, which are resulting from specification, implementation or realization errors, failure during operation period, reasonably foreseeable operational errors [and/or] reasonably foreseeable misuse.*" This definition concerns, in fact, the entire life cycle of a system. Safety control has to be effective during the preliminary phase of the system design (in particular, hazard analysis and risk assessment), during development (functional safety requirement allocation for hardware and software, and system evaluation) and even during operation services and decommissioning (verification that assumptions made during safety assessment and hazard analysis are still present). Once the function of a system has been specified, the safety process dictates that it goes over an established list of driving situations and their corresponding malfunctions and, for each one of them, gives the safety functions that are specified to avoid such situations as well as how to maintain the vehicle in a safe mode. Each of these situations is characterized by the frequency of its occurrences, the severity of the damage and the controllability of the situation by a driver. The system is characterized according to these parameters by a so-called Automotive Safety Integrity Level (ASIL). The format definition of the safety properties associated to each ASIL is not known at the present time. If we refer to the generic standard IEC 61508 [**?**], each SIL is defined by two kinds of safety properties: functional requirements, e. g. no erroneous signals are produced by an ECU, and safety integrity attributes, e. g. the probability of dangerous failure occurrences per hour has to be less than a given threshold (e.g. less than 10^{-8}.) Throughout the development of the system that realizes a function, it must be verified that this system ensures all the properties required by the SIL level assigned to the function. Verification activities are based, for example, on Failure Mode and Effect Analysis (FMEA), Fault or Event

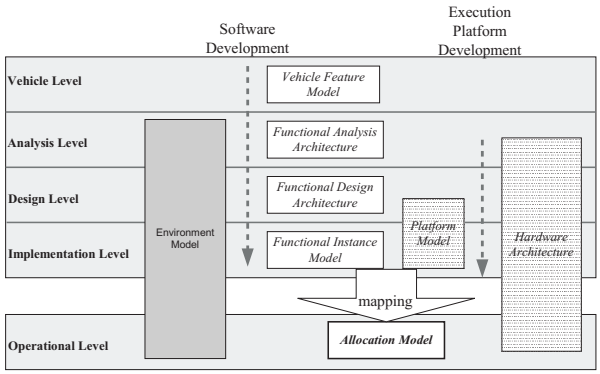

Figure 1. The abstraction levels and the system views in EAST-ADL

Tree Analysis, etc. completed by several techniques that could depend on the development process stage (formal methods and model checking, performance evaluation, schedulability and timing analysis, probability, Hardware In the Loop, System In the Loop, etc.)

IV. Conclusions

Nowadays, for any activity in our everyday life, we are likely to use products or services whose behavior is governed by computer-based systems, also called embedded systems. This evolution also affects the automotive industry. Several computers are embedded in todays vehicles and ensure functions that are vehicle centric, such as motor control, braking assistance, etc., as well as passenger centric such as entertainment, seat control, etc. This presentation has shown why this evolution is inescapable and has outlined the main thrusts of this development. First, state regulations, such as controlling exhaust emissions or mandatory active safety equipments (e.g. airbags), which impose embedding complex control laws that can only be achieved with computer-based systems. Secondly, customers are asking for more comfortable, easy-to-drive and safe cars and carmakers aim for launching new innovative products; both are costly. Today's advancing software technology appears to be a good trade-off between cost and product development, and therefore facilitates the introduction of new services in cars. In order to identify the requirements applied to embedded electronic systems, we presented a classification of these systems according to well identified functional domains. The pressure of these requirements affects the technological solutions in terms of hardware components as well as in terms of software development. Finally, the economical constraints push for the emergence of standards easing hardware / software independence, and consequently an efficient collaborative development process of embedded electronic architectures and the reuse of hardware and software entities. For example, at the present time, the CAN network is predominant in the interconnection of the ECUs (Electronic Control Unit). However, due to the increase in exchanges between ECUs, other solutions are emerging (e.g. the FlexRay network, the integration of mechatronic systems deployed on hierarchical distributed architecture, etc.). The growing complexity of the software embedded in a car reflects a well-mastered development process and efficient methods for the verification that the required properties (functional and non-functional) are met. Autonomous and automated road vehicles, communicating cars, and integrated traffic solutions are keywords for the vehicle of the future. These trends target controlling motorised traffic, decreasing congestion and pollution, and increasing safety and quality of living . In such a scenario, the development of a vehicle cannot be considered separately, but must be seen as a part of a complex system. Furthermore, the next standard OSI 26262, and those that are already being applied for road traffic, form another strong argument for solid, structured design methods. Thanks to international initiatives, such as AUTOSAR, the concepts of MBD (Model Based Development), MDD (Model Driven Development), and CBSE (Component Based Software Engineering) are penetrating the culture of automotive system designers. This will be possible as soon as tools supporting these concepts, and suited to the automotive industry, reach a higher maturity level.

V. References